Approaches to Middleware

406.306 Management Information Systems

Jonghun Park

jonghun@snu.ac.kr

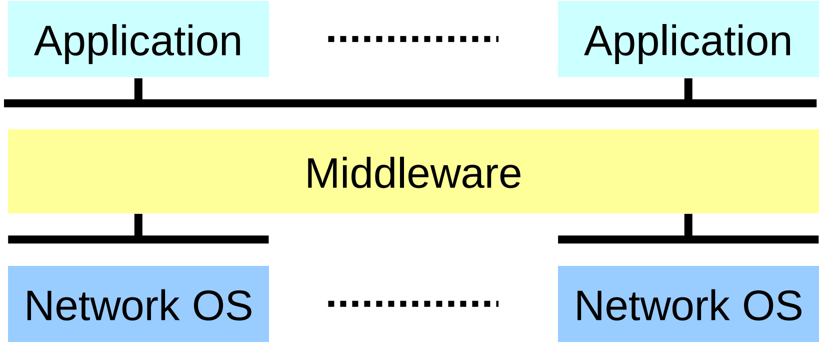
Dept. of Industrial Engineering

Seoul National University

9/20/2007

What is Middleware?

- connectivity software that is designed to help manage the **complexity** and **heterogeneity** inherent in distributed systems
- specifically, a layer of enabling SW services that allow application elements to interoperate across network links, despite differences in underlying comm. protocols, system architectures, OSs, DBs, and other application services
- builds a bridge between different systems by enabling communication and transfer of data
- applied to manage disparate applications both within one organizations and between various independent organizations
- customized software vs. standard package



Advantages of Middleware

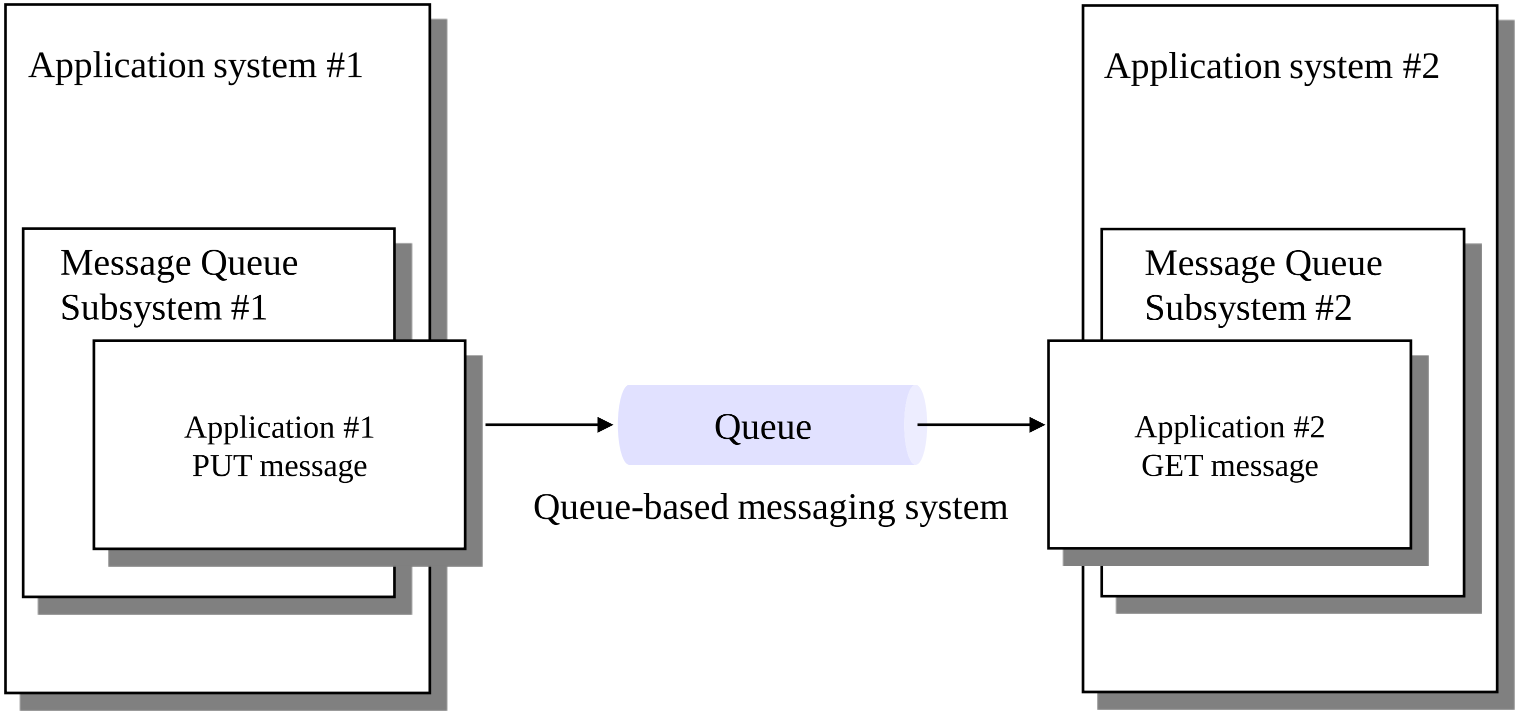
- Locate applications transparently across the network
  - transparency w.r.t. location, concurrency, replication, and failure
- Shield software developers from low-level details
- Provide a consistent set of higher level network oriented abstractions
- Leverage previous developments and reuse them
- Provide a wide array of services
- Scale up in capacity without losing function

basic messaging models

- Synchronous invocation
  - request / response
  - blocking, tightly coupled
- Asynchronous invocation
  - send and forget
  - nonblocking, loosely coupled
  - communication by sending messages that consist of header, properties, and payload (body)
- Asynchronous invocation with immediate acknowledgment
  - the invoked application would return an ack right away if the request is considered valid and then continue with its main computation
  - the calling computation would register a callback or poll in order to receive the ultimate result
- Execution is best effort, at most once
  - More than once is OK for idempotent operations, not otherwise
- Analogies in real-life: calling a travel agent

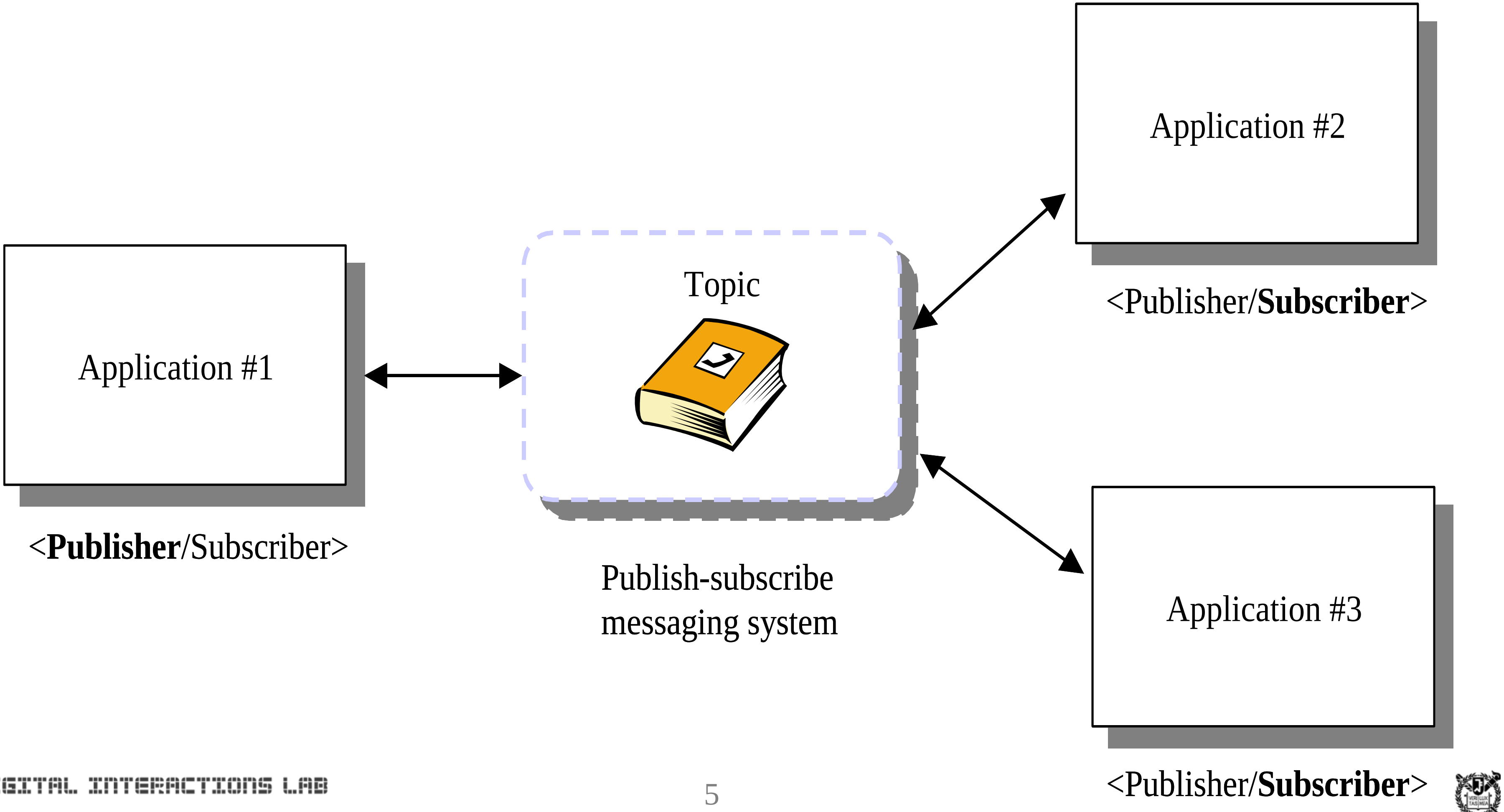
Store and Forward

- messages are placed on a message queue by the sending application and retrieved by the receiving application as needed
- typical of many-to-one messaging paradigm


Application system #1
Message Queue Subsystem #1
Application #1 PUT message
Queue
Queue-based messaging system
Application system #2
Message Queue Subsystem #2
Application #2 GET message

Publish and Subscribe

- application that produces information publishes it and all other applications that need this type of information, subscribe to it
- typical of many-to-many messaging paradigm

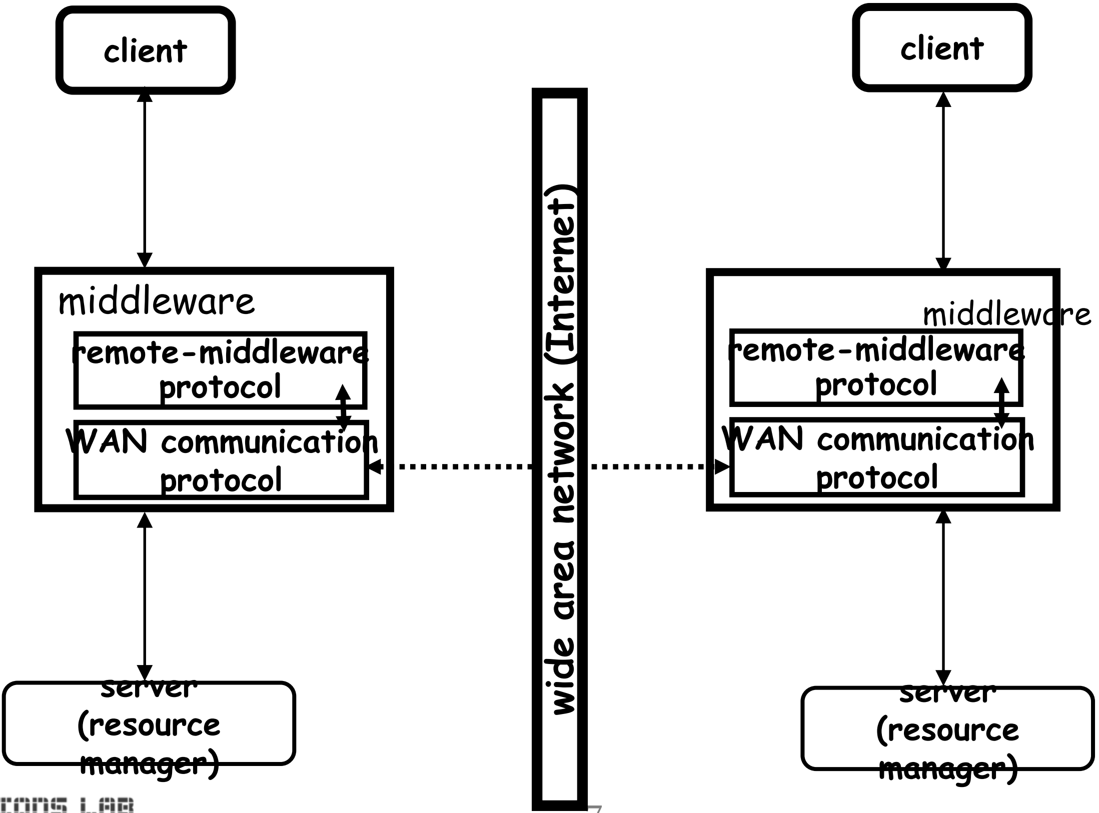

Application #1
<**Publisher**/Subscriber>
Topic
Publish-subscribe messaging system
Application #2
<Publisher/**Subscriber**>
Application #3
<Publisher/**Subscriber**>

Implications of Asynchronous Communication

- No longer a single thread of execution
  - improve performance
- Results arrive via a callback
  - enables the caller to perform other tasks and be notified when the result is available
- Asynchronous applications can execute in any order
  - caller must be able to determine which result came from which client application and combine the results together

Middleware extensions

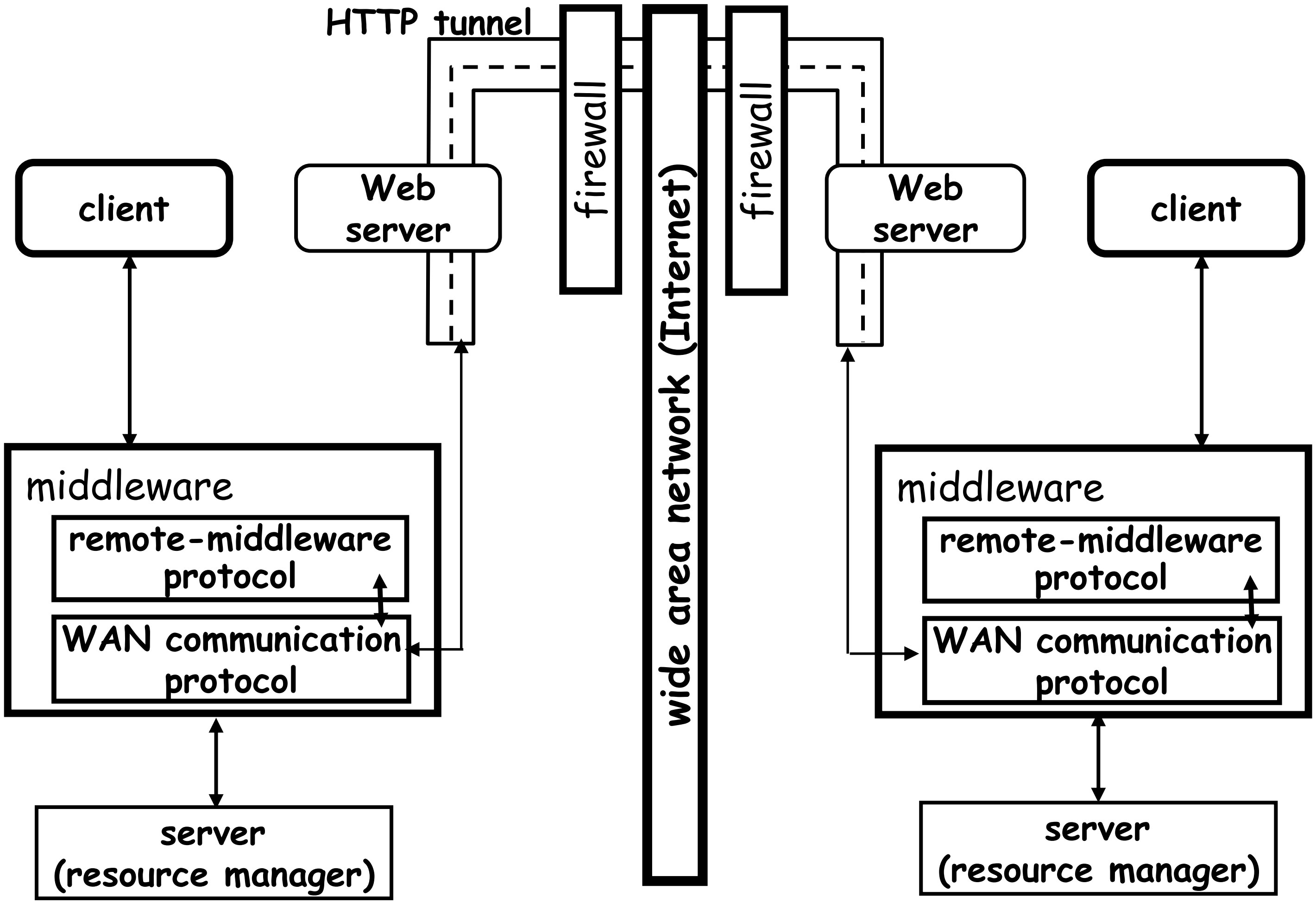
- To allow existing platforms to interact through the Internet, conventional middleware platforms were simply extended to support the Internet as one more access channel
  - requires the ability to invoke services residing in a different company
- Example: B2B transactions
- RMI, RPC, CORBA’s Inter-ORB protocol (GIOP)


client
middleware
remote-middleware protocol
WAN communication protocol
server (resource manager)
wide area network (Internet)
client
middleware
remote-middleware protocol
WAN communication protocol
server (resource manager)

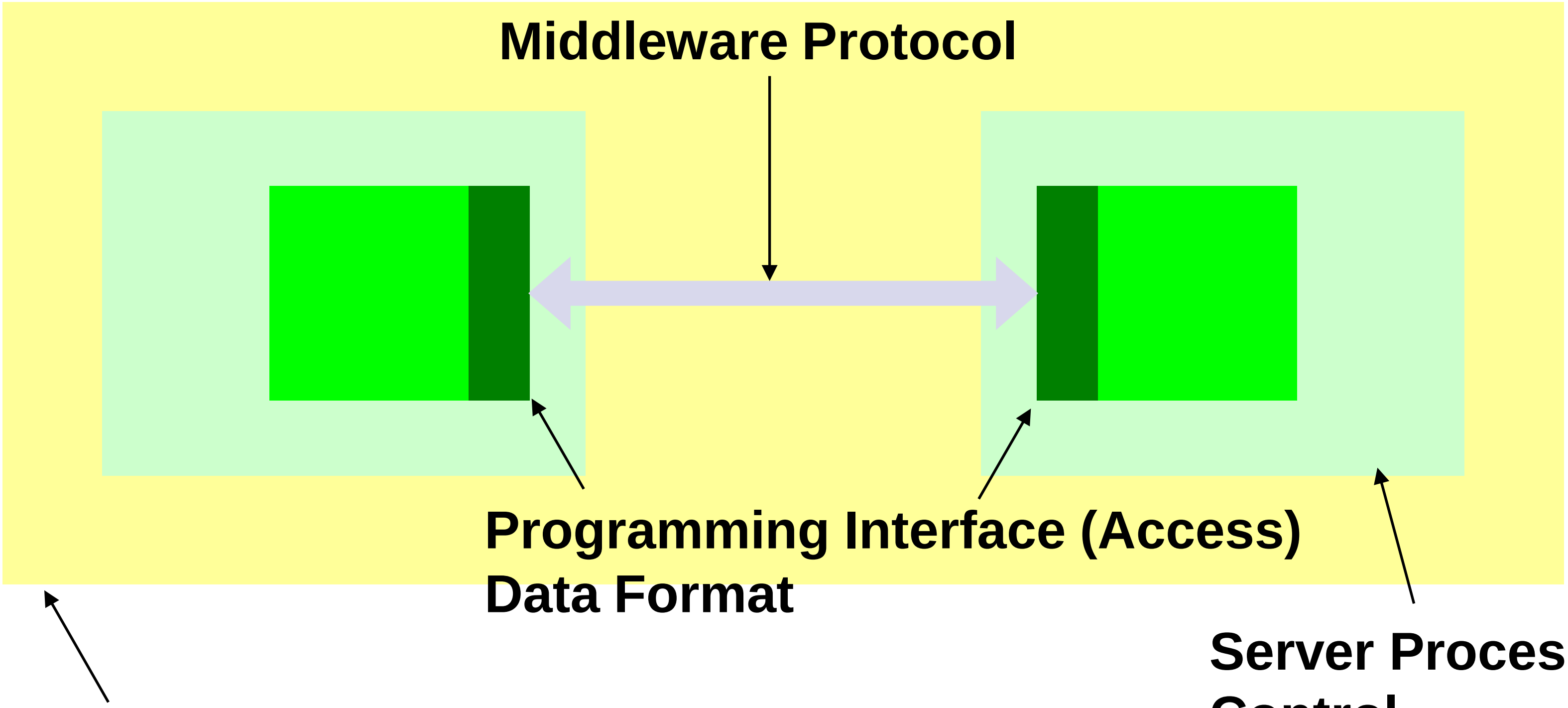
Middleware extensions

- Problems
  - Firewalls
    - No direct communication between the systems to be integrated is generally possible
    - Parties outside the firewall are not trusted
  - Agreement on the interface definitions and data formats
  - Directory server
- Tunneling
  - Protocols which would be blocked by the firewall are hidden under protocols that are accepted by the firewall
  - A call in one protocol that cannot get through the firewall is encapsulated within a call of another protocol that can get through the firewall
  - requires an intermediary conversion into HTML or XML document, sending the document using HTTP, and extracting the message from the document once it reaches the recipient
  - e.g., tunneling through HTTP or SSH, SOAP tunneling of RPC over HTTP

Tunneling


HTTP tunnel
client
Web server
firewall
wide area network (Internet)
firewall
Web server
client
middleware
remote-middleware protocol
WAN communication protocol
server (resource manager)
middleware
remote-middleware protocol
WAN communication protocol
server (resource manager)

Middleware elements



Communication Protocol
Middleware Protocol

Programming Interface (Access)
Data Format

Server Process
Control

Naming / Directory Services
Persistence Services
Security
Administration

Middleware as infrastructure

- Middleware is a very complex software system
- Requires basic infrastructure such as
  - IDL
  - IDL compiler
  - Libraries
  - Run-time support
  - Authentication
  - Addressing
  - Naming
  - Low level protocol used
  - Multi-threading
  - Logging
  - Transactions
  - Asynchronous messaging
  - and many more

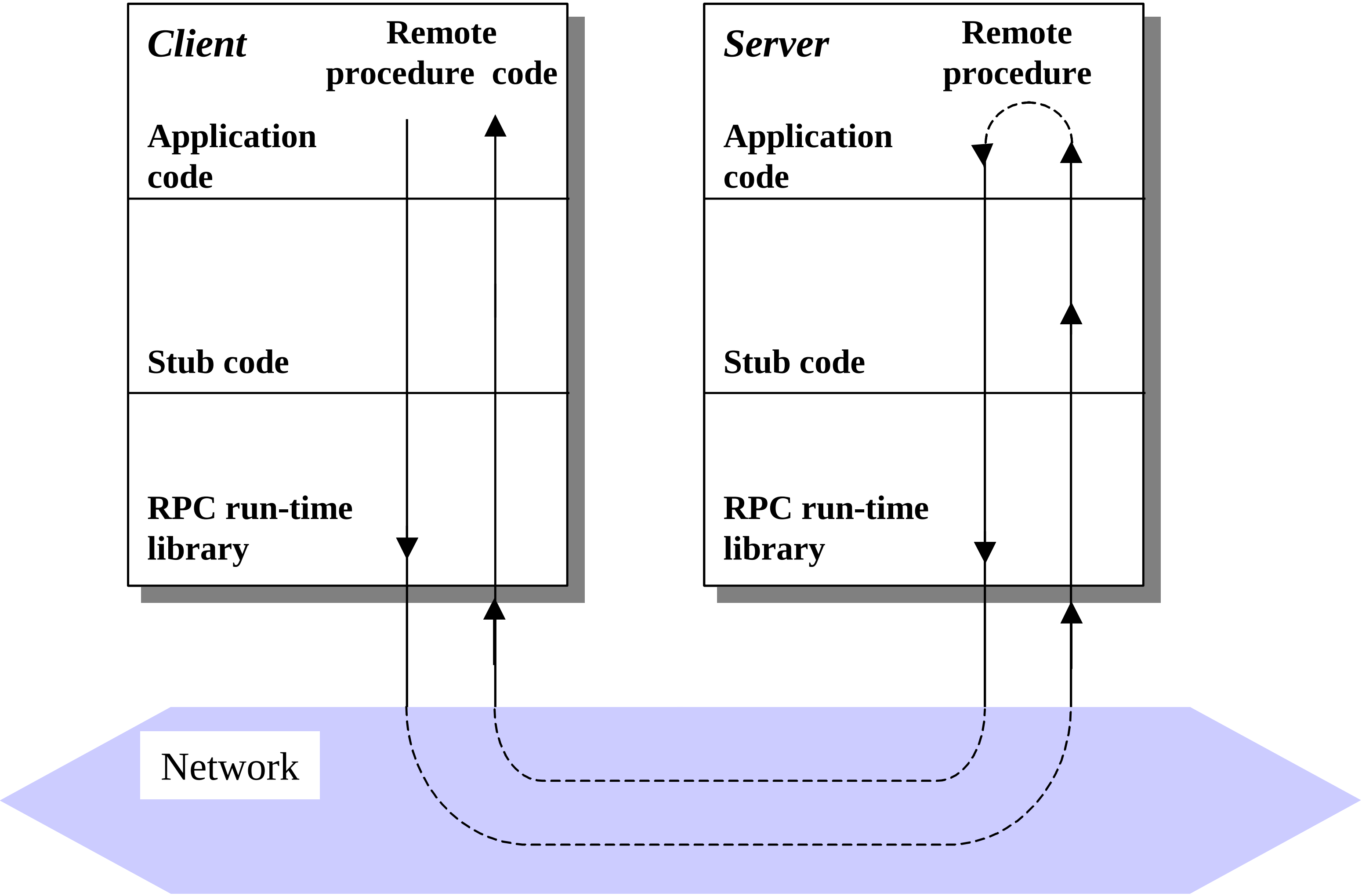
Types of middleware

- Remote Procedure Calls (RPCs)
  - provides the infrastructure necessary to transform procedure calls into remote procedure calls in a transparent manner
- Message oriented middleware
  - provides transactional access the queues, persistent queues, and a # of primitives for reading and writing to local and remote queues
  - e.g., WebSphere MQ Family (IBM), MSMQ (Microsoft)
- Data-access middleware
- Transaction-oriented middleware
  - can be seen as RPC with transactional capabilities
- Object Request Brokers (ORBs)
  - supports the invocation of remote objects, thereby leading to object brokers
  - e.g., RMI / Jini (Sun), CORBA (OMG)

Remote Procedure Calls

- RPC is the middleware mechanism used to invoke a procedure that is located on a remote system, and the results are returned
- With this type of middleware the application elements communicate with each other **synchronously**, meaning they use a request/wait-for-reply model of communication
- simplest type of middleware
- work well for smaller, simple applications where communication is primarily point-to-point
- do not scale well to large, mission-critical applications
- e.g., opening a remote folder at windows

Remote Procedure Calls



Client

Remote procedure code

Application code

Stub code

RPC run-time library

Server

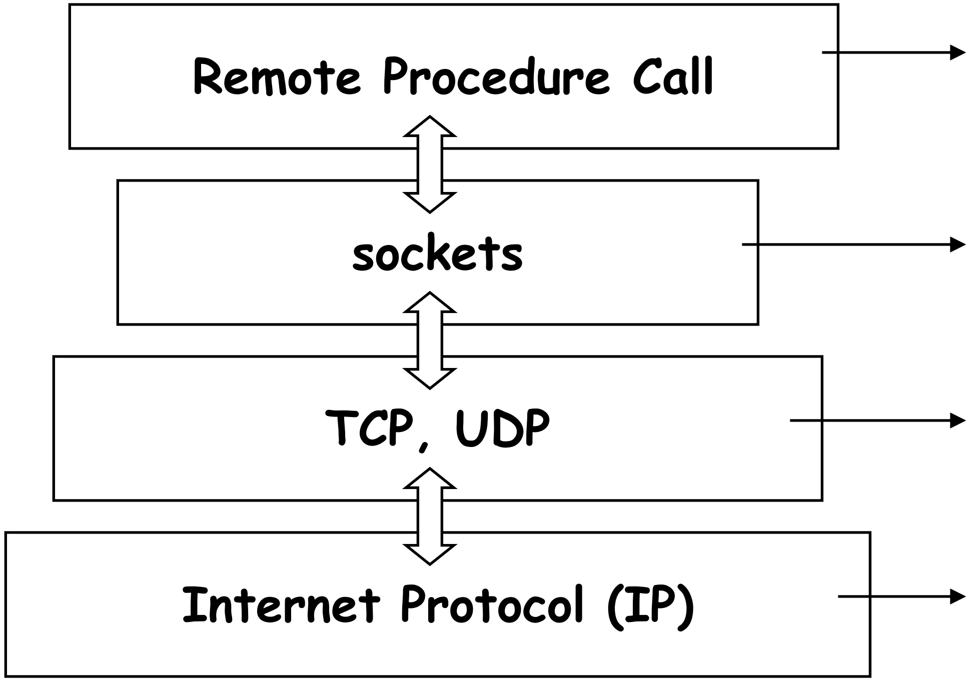
Remote procedure

Application code

Stub code

RPC run-time library

RPC as a programming abstraction



Remote Procedure Call:
hides communication details behind a procedure call and helps bridge heterogeneous platforms

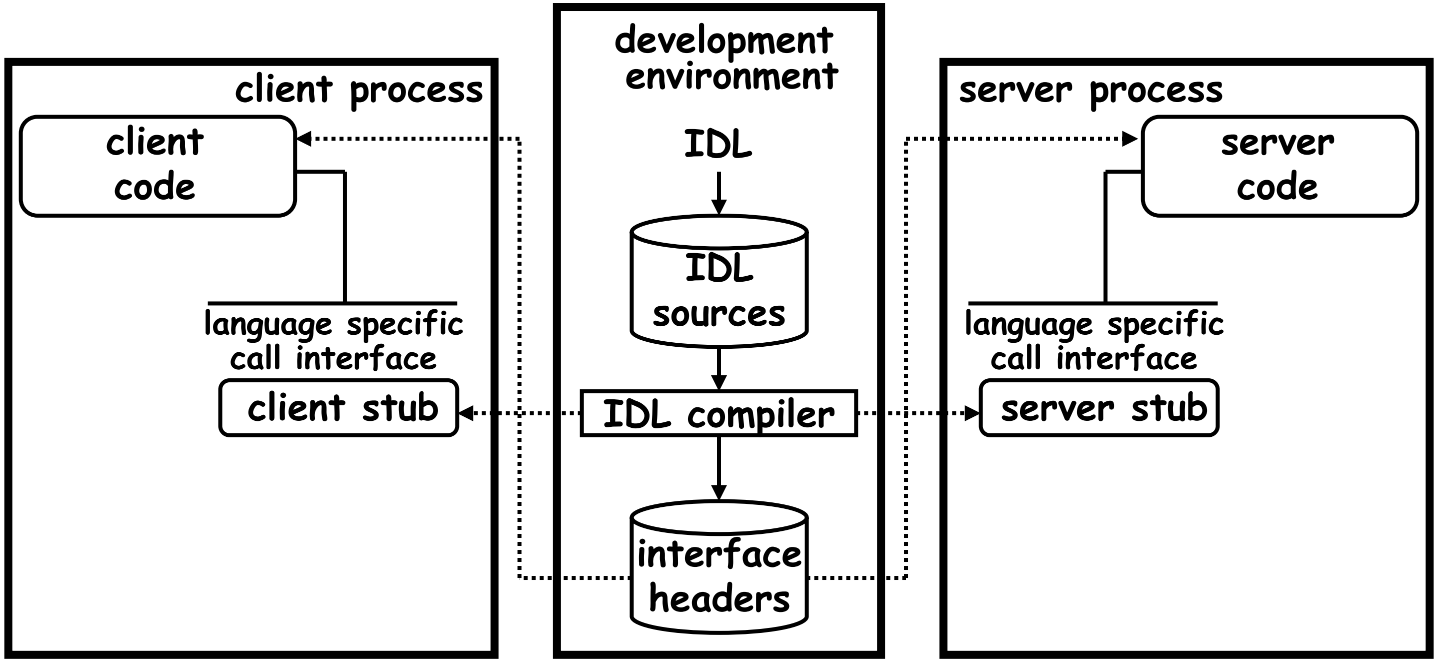
sockets:
operating system level interface to the underlying communication protocols

TCP, UDP:
User Datagram Protocol (UDP) transports data packets without guarantees
Transmission Control Protocol (TCP) verifies correct delivery of data streams

Internet Protocol (IP):
moves a packet of data from one node to another

RPC: How RPC works

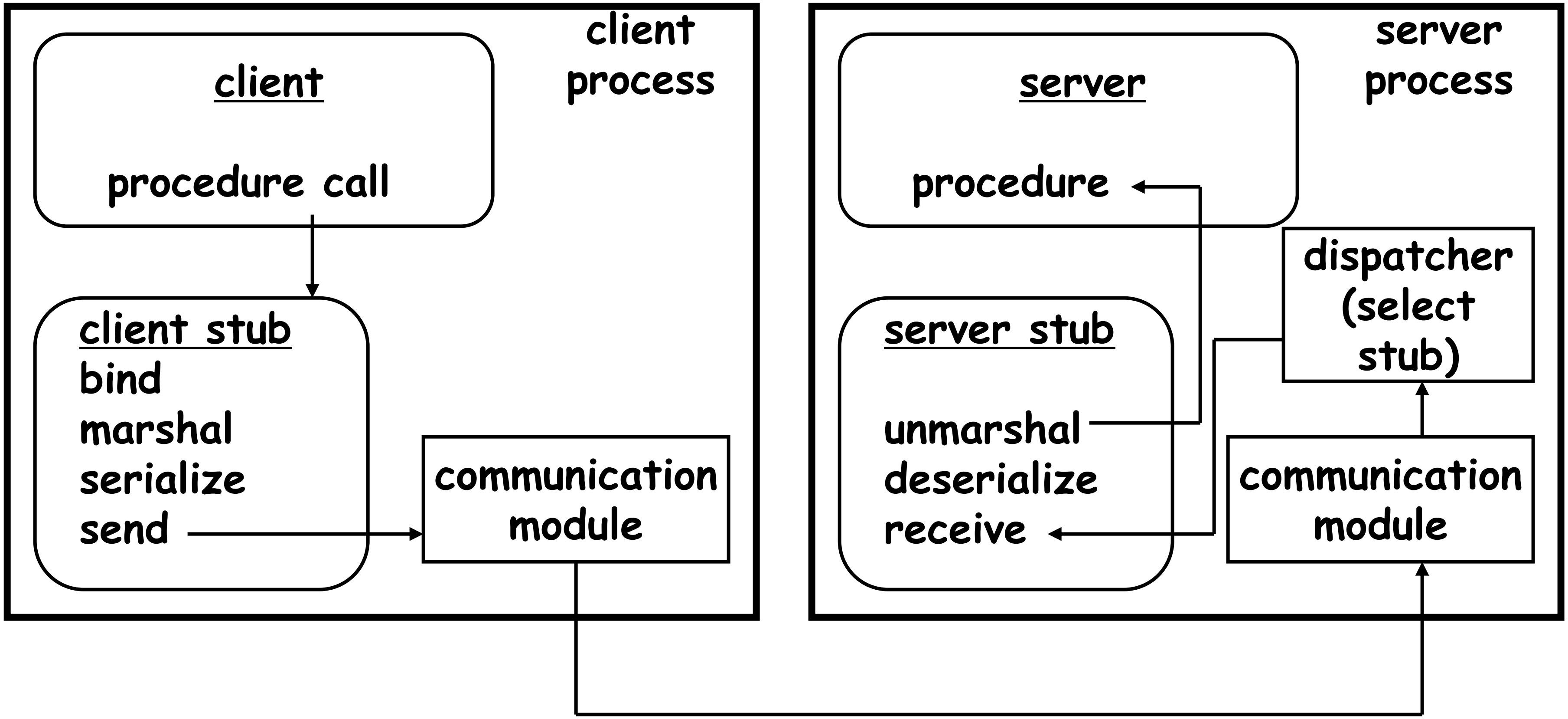
- IDL: provides an abstract representation of the procedure in terms of what parameters it takes as input and what parameters it returns as a response



RPC: How RPC works

- Client stubs:
  - A piece of code to be compiled and linked with the client
  - carry out binding, marshaling, serializing, communicating with the server, getting a response, forwarding the response
  - proxy for the actual procedure implemented at the server
- Server stubs:
  - similar to the client stub except that it implements the server side of invocation
  - receiving the invocation from the client stub, deserializing and unmarshaling the call, invoking the actual procedure, forwarding the results to the client stub
- Code templates and references:
  - IDL compiler generates necessary header files and templates with the basic code

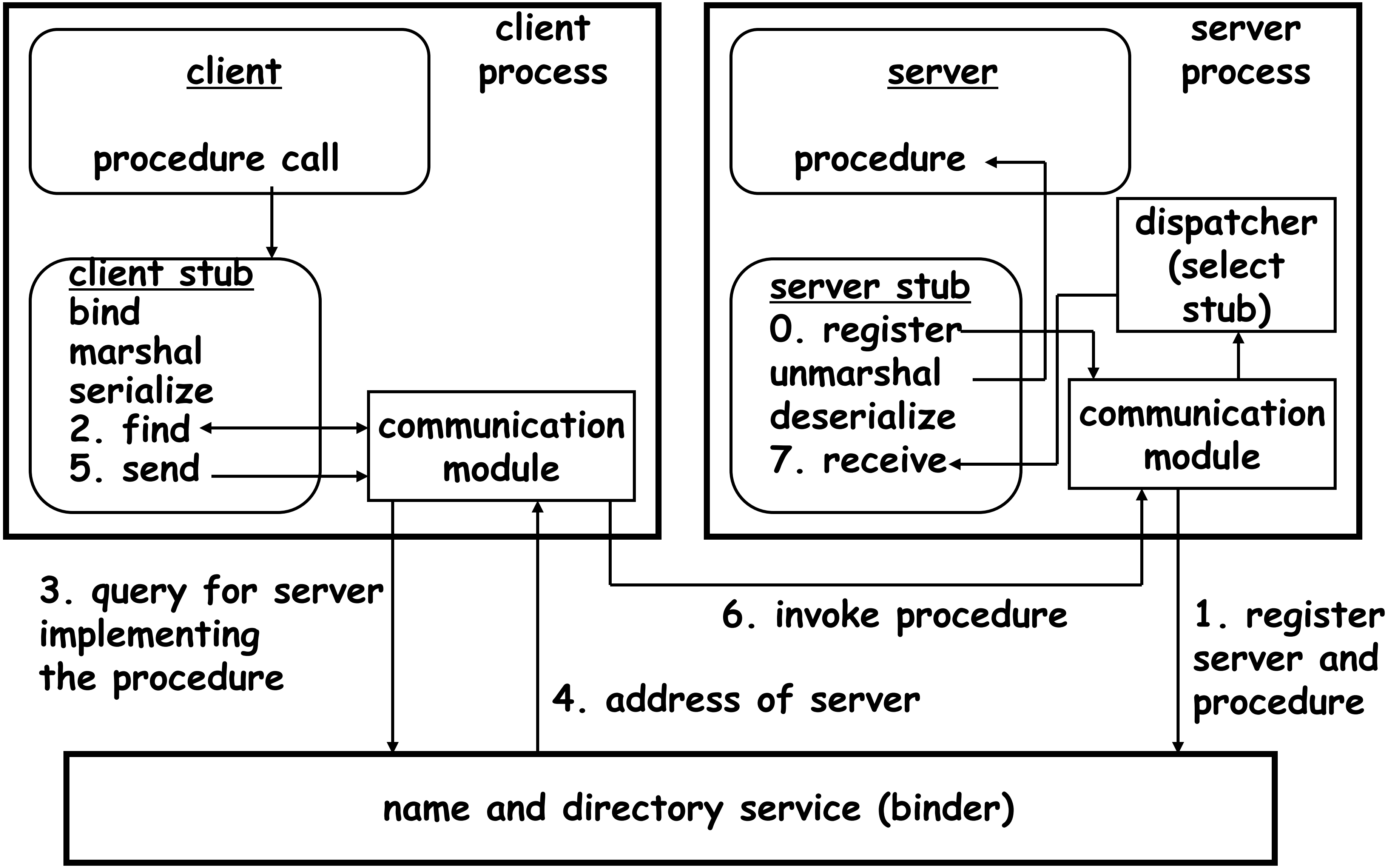
Basic functioning of RPC



Binding in RPC

- Binding is the process whereby the client creates a local handle to a given server in order to invoke a remote procedure
- Static binding
  - the client stub is hardcoded to already contain the handle of the server where the procedure resides
  - simple, efficient, but tightly coupled (e.g., server failure, server location change)
- Dynamic binding
  - enables clients to use a specialized service to locate appropriate servers
  - when the client invokes a remote procedure, the client stub asks the directory server for a suitable server to execute that procedure
  - adds a layer of indirection to gain flexibility at the cost of performance

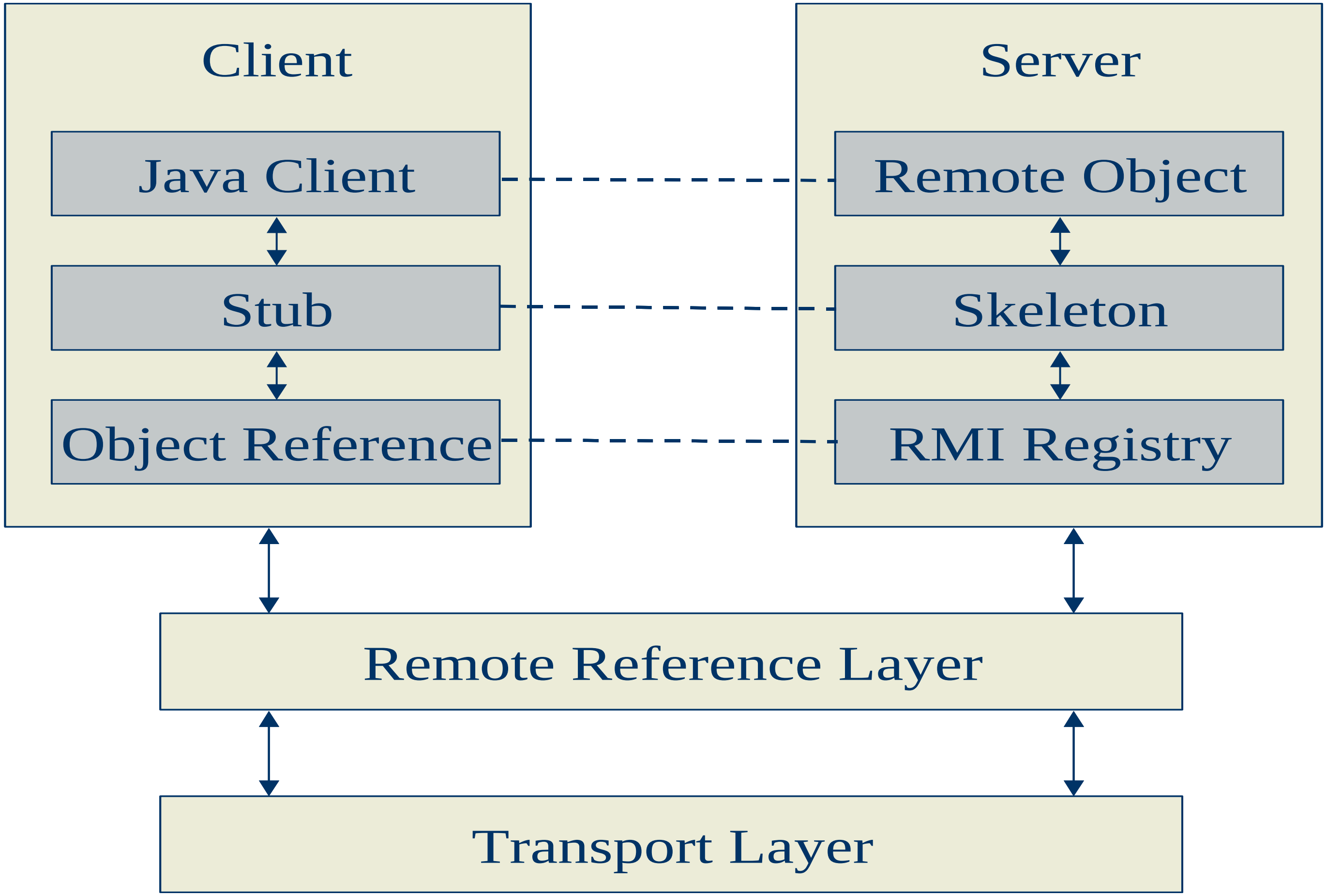
Dynamic binding



Remote Method Invocation

- 2 separate programs, server and client
- provides a simple and direct model for distributed computation with Java objects on the basis of the RPC mechanism
- server application creates some remote objects, makes references to them accessible, and waits for clients to invoke methods on these remote objects
- Client → Stub
- Server → Skeleton

The Java RMI



Client

Java Client

Stub

Object Reference

Server

Remote Object

Skeleton

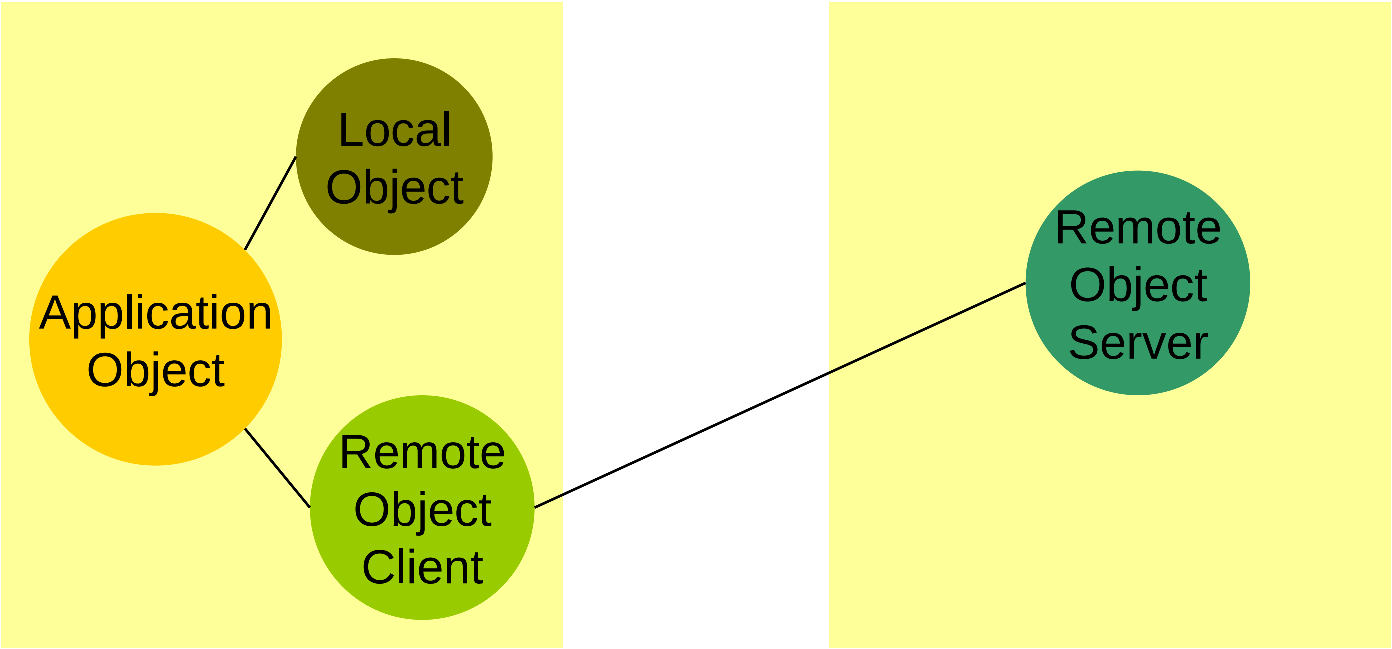
RMI Registry

Remote Reference Layer

Transport Layer

Java RMI

- Foundation for J2EE, Jini, and other Java based distributed-object technologies
- RMI enables cross-JVM, cross-machine method calls
- advantages over RPC: e.g., task server
- http://java.sun.com/products/jdk/rmi/



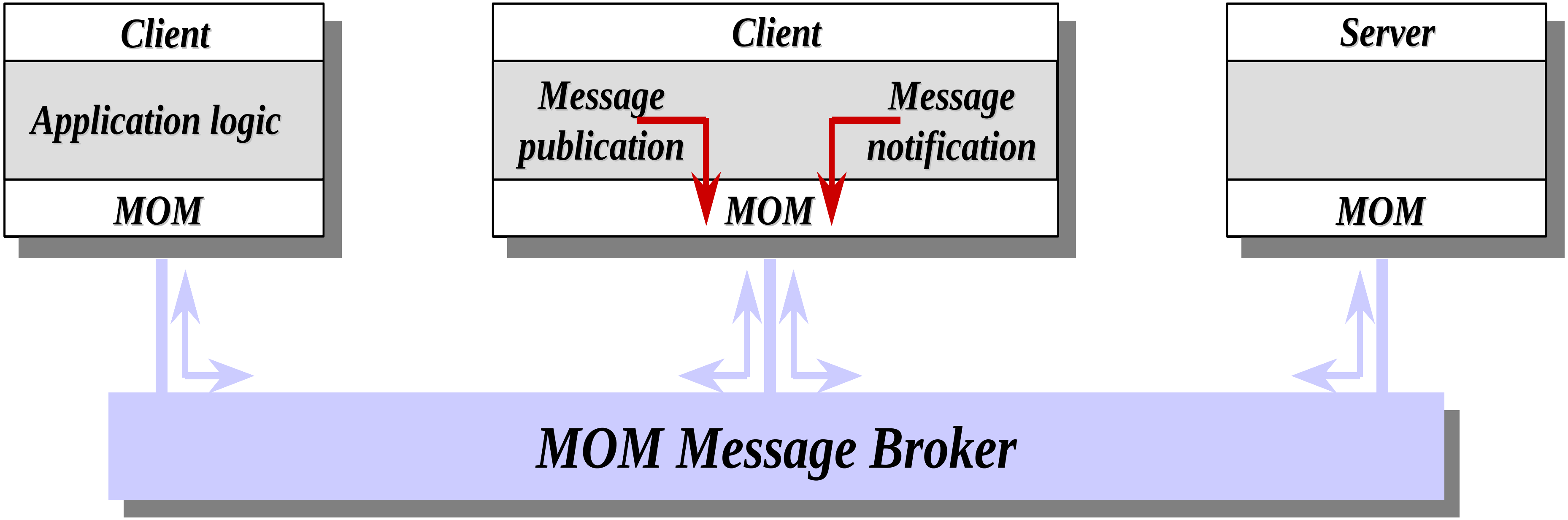
More on Java RMI

- More interesting compute engine implementation based on RMI can be found at http://java.sun.com/docs/books/tutorial/rmi/index.html
- RMI activation framework

Message-Oriented Middleware (MOM)

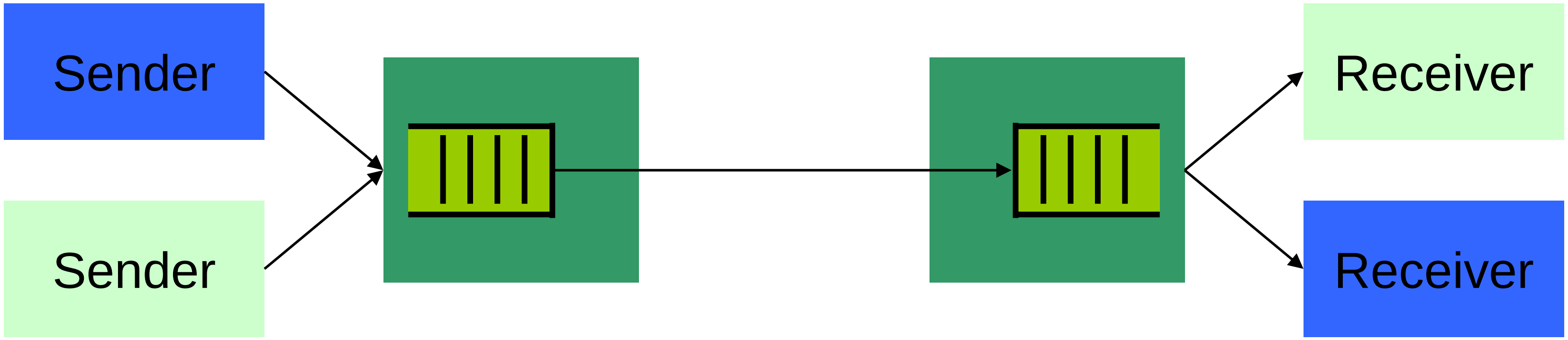
- Message Oriented Middleware (MOM) is the back-bone infrastructure that is responsible for relaying data from one application to another by putting it in a uniform message format.
- similar to email system
- loosely coupled
- Features that make the MOM particularly **attractive** when integrating applications :
  - applications need to automatically or periodically pass data to each other
  - integration nature is event driven
  - Prioritization of requests
  - Load balancing
  - Persistent messaging
- disadvantage: overloading due to the temporary storage

MOM


Client
Application logic
MOM
Client
Message publication
Message notification
MOM
Server
MOM
MOM Message Broker

Message-oriented middleware

- Today most large integration efforts are done using MOM
- Example: IBM WebSphere MQ, Microsoft's MSMQ, WebMethods Enterprise
- More robust to failures w.r.t. RPC or object brokers
- Provides persistent communication between processes through intermediate-term storage capacity for messages
- Does not require either the sender or receiver to be active during message transmission
  - Loosely-coupled, asynchronous
  - The sender and receiver are completely independent
- Transactional queues
  - guarantees exactly once semantics

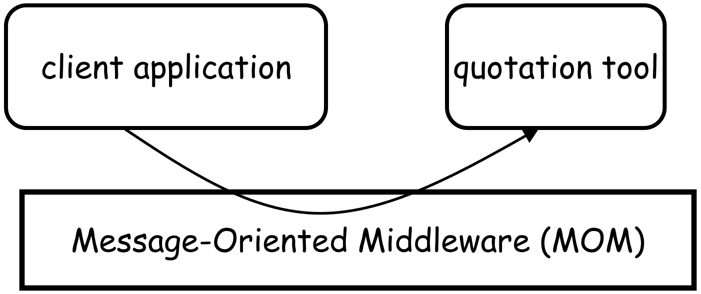


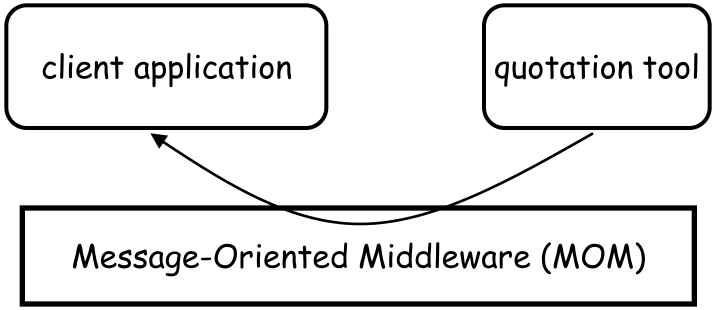
Message-based interoperability

- refers to an interaction paradigm where clients and service providers communicate by exchanging messages
- Message: a structured data set, typically characterized by a type and a set of <name, value> pairs that constitute the message parameters
- Most product use XML types

```
Message : quoteRequest {
 QuoteReferenceNumber: 325
 Customer: Acme,INC
 Item:#115 (Ball-point pen, blue)
 Quantity: 1200
 RequestedDeliveryDate: Mar 16,2003
 DeliveryAddress: Palo Alto, CA
}
```

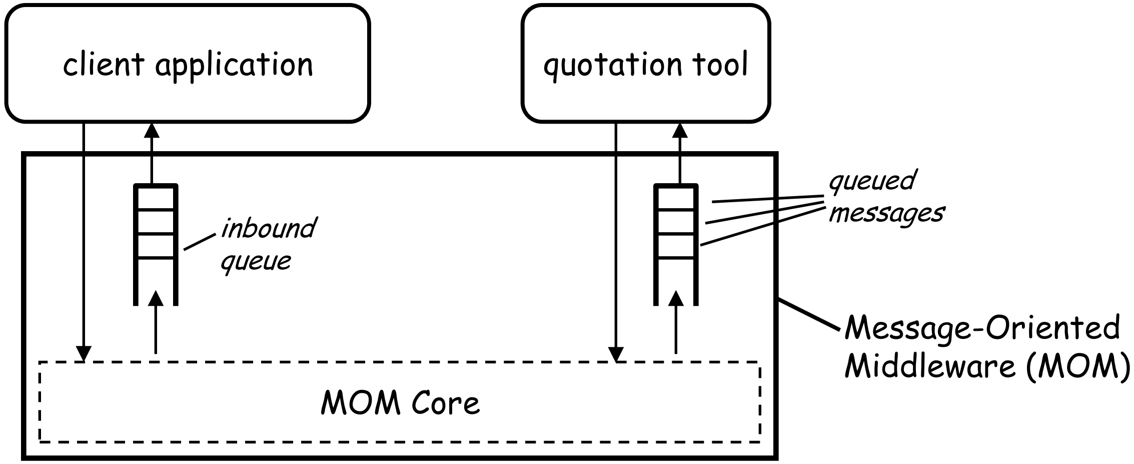
```
Message: quote {
 QuoteReferenceNumber: 325
 ExpectedDeliveryDate: Mar 12, 2003
 Price:1200$
}
```





Message queues

- considerably simplifies the development of interoperable applications and provides support for managing errors or system failures
- messages sent by MOM clients are placed into a queue, typically identified by a name, and possibly bound to a specific intended recipient
- whenever the recipient is ready to process a new message, it invokes the suitable MOM function to retrieve the first message in the queue
- queued message may have an associated expiration or retrieval



Interacting with a message queueing system

- Queueing systems provide an API that can be invoked to send messages or to wait for and receive messages
- Sending a message is typically a nonblocking operation
- Receiving a message is instead often a blocking operation, where the receiving object “listens” for messages and process them as they arrive, typically by activating a new dedicated thread, while the “main” thread goes back to listen for the next message
- Recipients can also retrieve messages in a nonblocking fashion by providing a callback function that is invoked by the MOM each time a message arrives
- JMS: an industry standard API for interacting with MOMs
- Open source: JORAM, JBossMQ

Integration Brokers

- an application-to-application middleware service that is capable of one-to-many, many-to-one, and many-to-many message distribution
- a software hub that records and manages the contracts between publishers and subscribers of messages
- The integration broker transforms application specific messages into commonly understood messages, e.g., between different XML schemas using eXtensible Stylesheet Language Transformations (XSLT)

Features of Integration Brokers

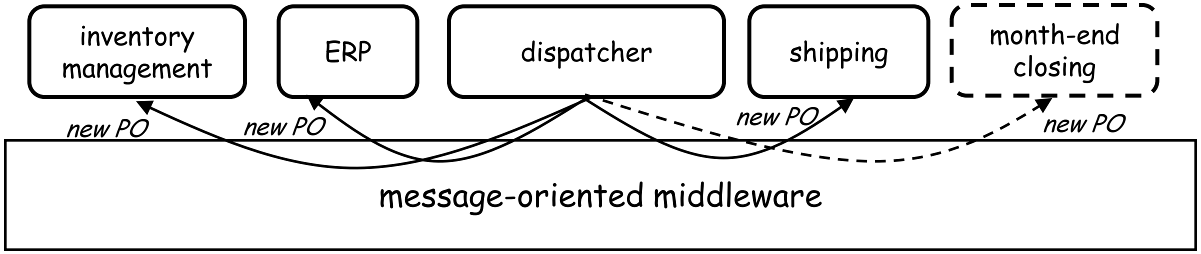
- Message transformation
- Business rules processing
- Routing services
- Directory services
- Adapter services
- Repository services
- Events and alerts

more on integration brokers

- Traditional RPC-based and MOM systems create point-to-point links between applications
- MOM
  - did not provide support for defining **sophisticated logic for routing messages**
  - did not help developers to cope with the heterogeneity
- Message brokers address this limitation by acting as a broker among system entities, thereby creating a “hub and spoke” communication infrastructure for integrating applications
  - provides routing, filtering, and processing logic for the messages as they move across the system
  - provides adapters that mask heterogeneity
- Commercial products
  - ActiveSoftware -> acquired by WebMethods
  - IBM WebSphere MQ
  - Tibco ActiveEnterprise
  - BEA WebLogic Integration

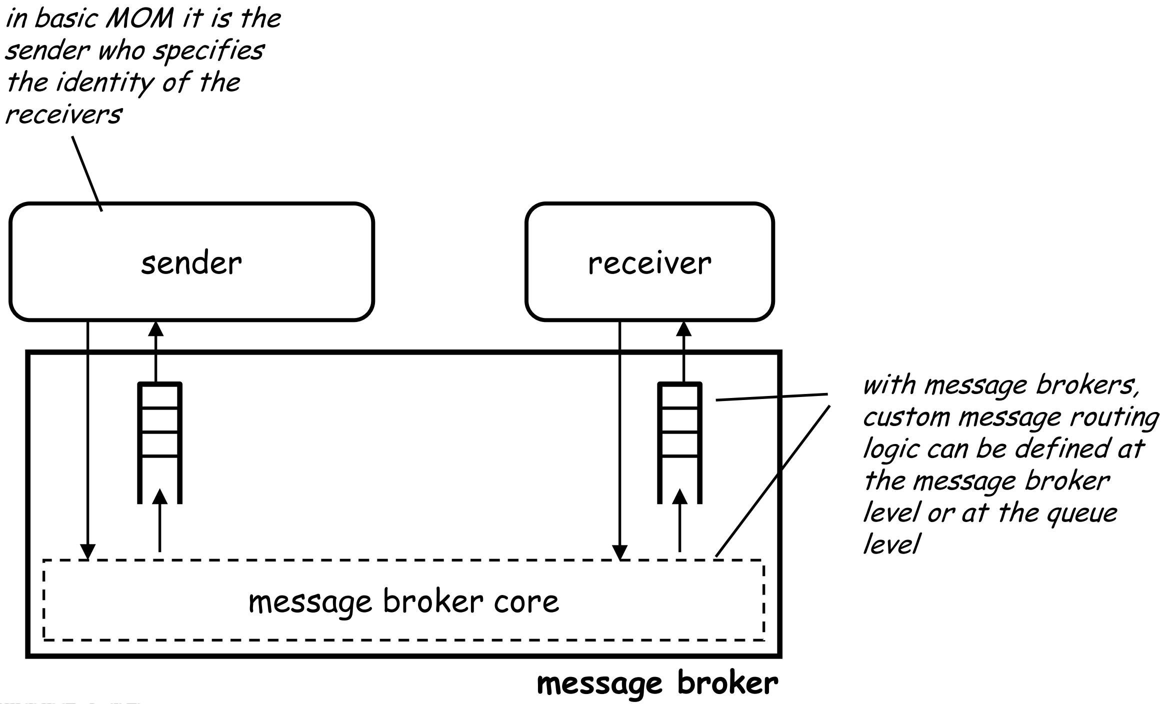
The need for integration brokers

- Example: Many different systems will need to process PO
  - Inventory management applications to check availability
  - ERP systems to manage payments
  - Shipping applications to arrange for delivery of goods
- With RPC or message-based interoperability, applications need to be changed if they need to interoperate with a new system



Extending basic MOM

- MOM
  - The responsibility for defining the receiver of a message lies with the sender -> becomes a complex problem as the # of senders and recipients grows
- integration brokers
  - factors the message routing logic out of the senders and placing it into the middleware
  - Users can define application logic that identifies, for each message, the queues to which it should be delivered
  - It is up to the message broker to identify the recipients by executing user-defined rules

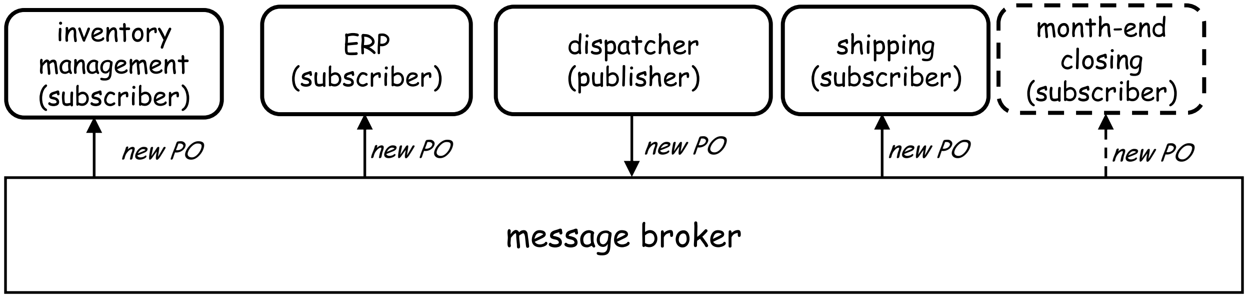

in basic MOM it is the sender who specifies the identity of the receivers
sender
receiver
with message brokers, custom message routing logic can be defined at the message broker level or at the queue level
message broker core
message broker

Extending basic MOM

- Routing logic can be based on the sender’s identity, on the message type, or on the message content
- integration brokers can decouple senders and receivers
  - Senders do not specify and are not aware of which applications will receive the messages they send
  - Receivers may or may not be aware of which applications are capable of sending messages to them
- Associating processing logic with queues
  - enables the content transformation rules: e.g., pound vs. kg
  - Problems
    - makes difficult to debug and maintain
    - with many logics, the overall latency and throughput is degraded
    - inability to handle large messages

Publish / subscribe interaction model

- Applications that send messages simply publish the messages to the middleware system that handles the interaction
- If an application is interested in receiving messages of a given type, then it must subscribe with the middleware
- Whenever a publisher sends a message of a given type, the middleware retrieves the list of all applications that subscribed to messages of that type, and delivers a copy of the message to each of them



Publish / subscribe interaction model

- Subscribers have two main ways to define the messages they are interested in receiving
  - message type: e.g., newPO, SupplyChain.newPO, SupplyChain.*
  - parameter-based: e.g., type="new PO" AND customer="ACME Co." AND quantity > 1200
- Virtually every message broker today supports the publish / subscribe interaction paradigm

Data-access Middleware

- Command Line Interfaces (CLIs):a common API that can manage access to different types of relational databases via a well-defined common interface
  - e.g., JDBC
- Native database middleware to access a particular database using only native mechanisms rather than a single multi-database API
- Database gateways (also known as SQL gateways) provide access to data that reside in different types of platforms

Transaction Oriented Middleware

- Transaction Processing Monitors provide the distributed client/server environment the capacity to efficiently and reliably develop, execute and manage transaction applications
- supports ACID properties
- TP monitor
  - provides the distributed client/server environment with capacity to efficiently and reliably develop, execute, and manage transaction applications
  - invented to run applications that server large numbers of clients
  - more intrusive than MOM: demand more modification of the applications themselves
- application server
  - offer an integrated development environment that allows enterprises to connect and manage front-office and back-office applications and combine them with web-enabled functionality
  - normally based on 3-tier model

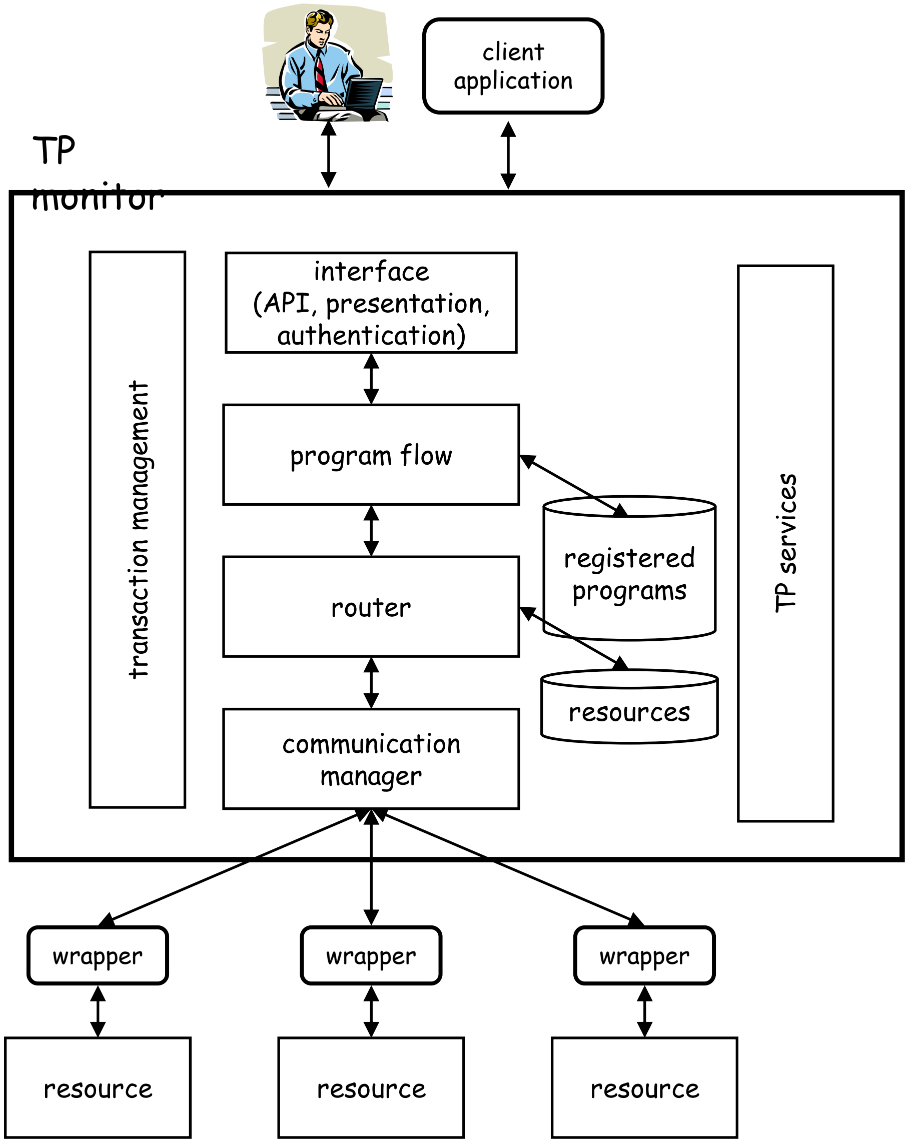
TP monitors: History

- TP monitors predate client/server architecture
- IBM’s CICS developed at the end of 1960s: still in use
- Originally designed to allow mainframes to support the efficient multiplexing of resources among as many concurrent users as possible
- Almost all commercial TP monitors became 3-tier systems
- TP-lite monitors: provide the core functionality of a TP monitor as an additional layer embedded in DBMS
- Examples
  - CICS: 1-tier
  - BEA’s Tuxedo: originally 2-tier queue-based system
  - Microsoft’s MTS

TP monitor

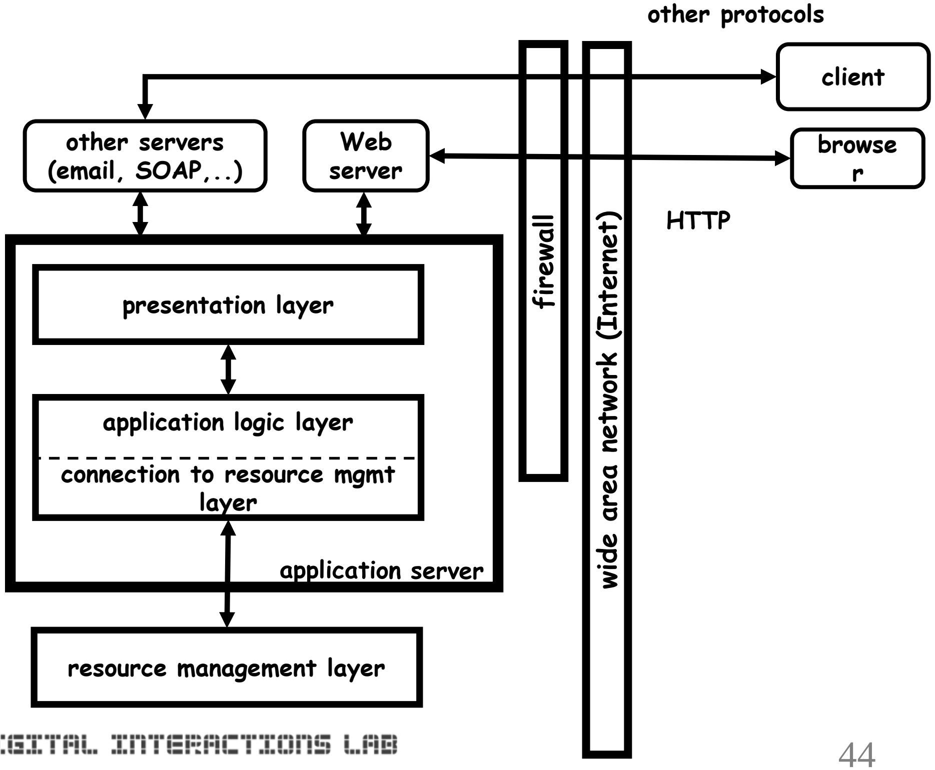
- provides transactional guarantees over all the resources that it controls
- provides concurrency control and recovery across processes
- intimately associated with application servers and provide some of the same kinds of functionality, such as hosting applications, managing threads and processes, and pooling connections to DBs

Architecture of a TP monitor

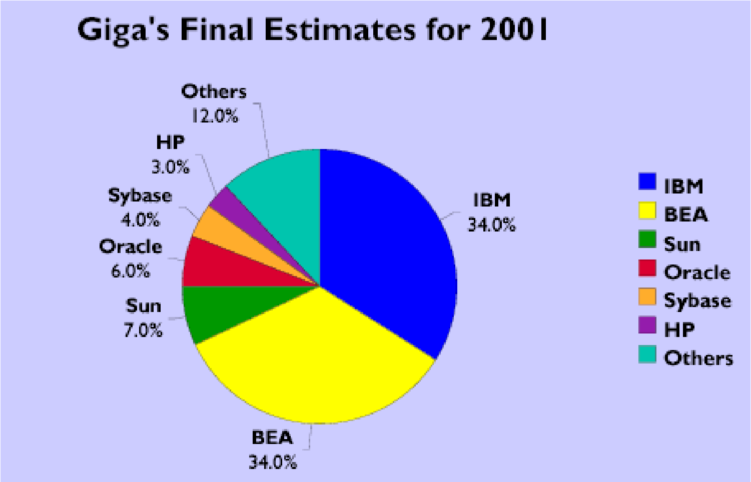


Application servers

- The increasing use of the web as a channel to access information systems forced middleware platforms to provide support for web access
- Difference from the conventional middleware: the incorporation of the web as a key access channel to the services implemented using the middleware
- BEA WebLogic, IBM WebSphere, Sun ONE, MS .NET, Oracle AS, JBoss, Sybase EAServer
- Increased flexibility, but cannot match the performance of TP monitors
- Web servers may or may not be included in a vendor’s offering



other protocols
client
other servers (email, SOAP,..)
Web server
browse r
HTTP
firewall
wide area network (Internet)
presentation layer
application logic layer
connection to resource mgmt layer
application server
resource management layer

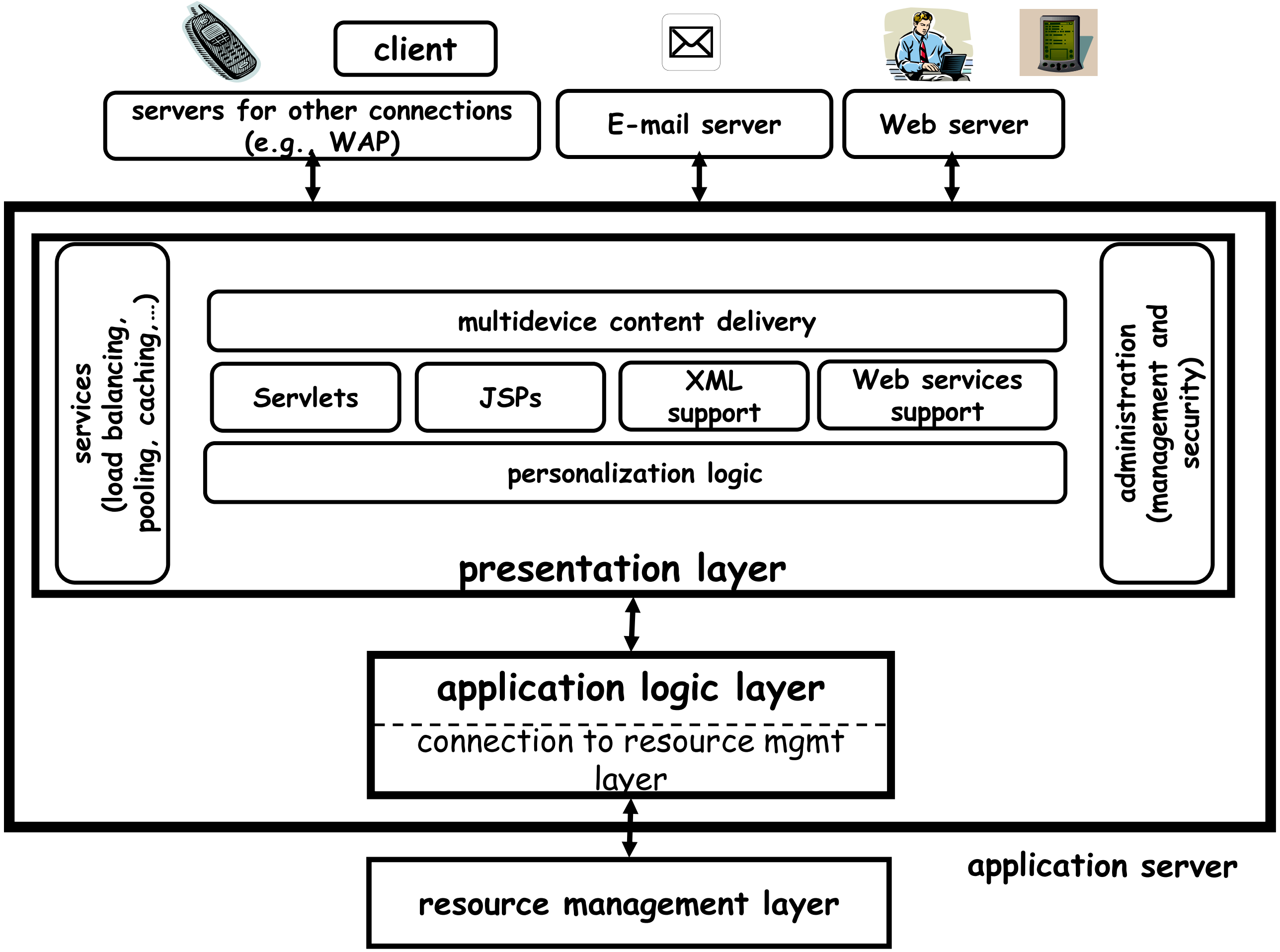


Objectives of application servers

- Execution of business logic
  - provides a platform of services to share and execute business logic components (e.g. EJB, servlets)
- High performance
  - connection pooling, multithreading, caching
- Scalability
  - clustering service, horizontal scalability
- High availability
  - eliminates single points of failure
- Security management
- Transaction management
- Systems management
  - promotes a distributed component-based computing model, supports SNMP to start, stop, and monitor business components
- Development tools and services

AS support for the presentation layer

- Clients: web browsers, applications, devices, e-mail programs, web services clients


client
servers for other connections (e.g., WAP)
E-mail server
Web server
multidevice content delivery
Servlets
JSPs
XML support
Web services support
personalization logic
services (load balancing, pooling, caching,...)
administration (management and security)
presentation layer
application logic layer
connection to resource mgmt layer
application server
resource management layer

Distributed Object Middleware

- Distributed objects are a development of RPCs that provide an additional layer of interoperability that abstracts the procedure call from the underlying platform and language
- examples
  - ORBs
  - EJB component model

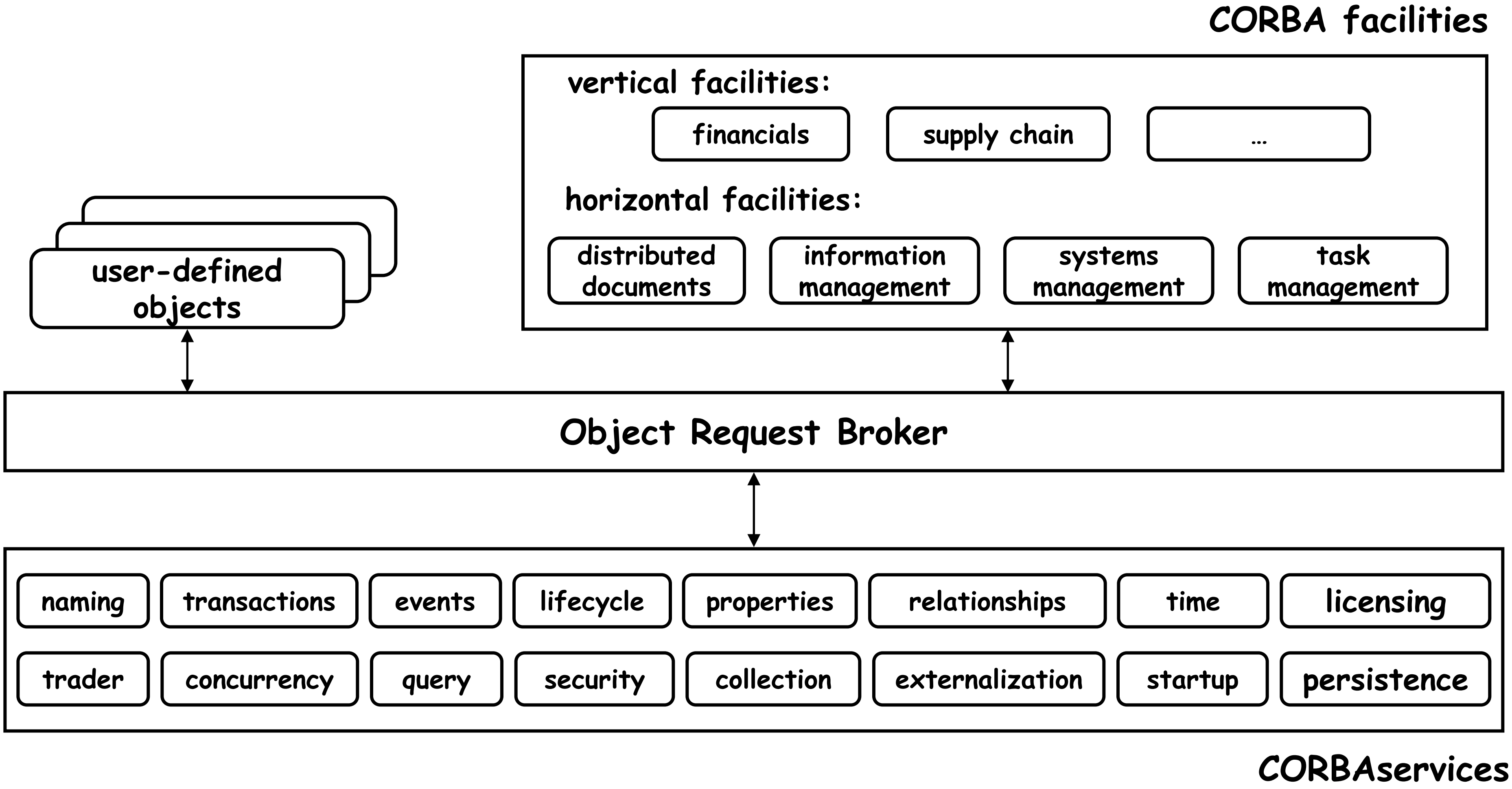
Object request brokers

- a distributed-object middleware technology that manages communication and data exchange between objects
- extends the RPC paradigm to the OO world and provide a number of services that simplify the development of distributed OO applications
- appeared at the beginning of the 1990s as the natural evolution of RPC to cope with object orientation
- CORBA: the best known example of object broker
  - developed in the early 1990s by OMG
  - offers a standardized specification of an object broker rather than a concrete implementation
  - enjoyed tremendous popularity in the mid- and late- 1990s
  - can perform dynamic service selection and invocation -> rarely used
- DCOM, COM+, .NET, J2EE

CORBA

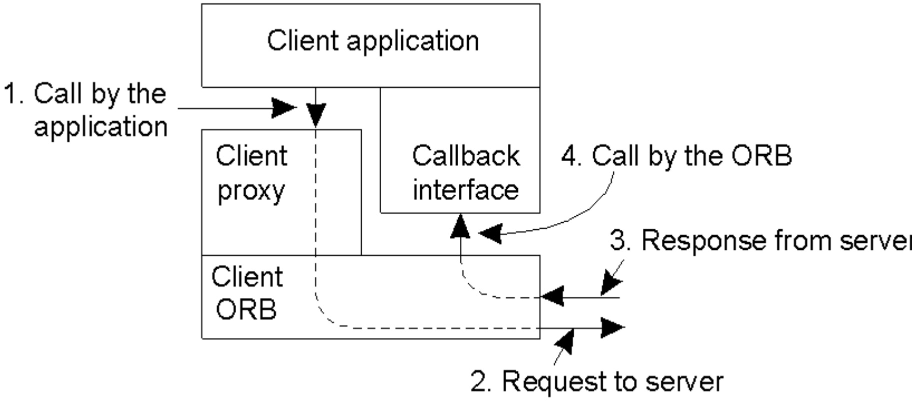
- Common Object Request Broker Architecture
- OMG (Object Management Group)
  - A nonprofit organization with over 800 members primarily from industry
- Quite popular in UNIX-based systems
- Specifications
  - http://www.omg.org
- Implementations
  - Orbix: http://www.iona.com (commercial)
  - VisiBroker-RT: http://www.borland.com (commercial)
  - MICO: http://www.mico.org (free)

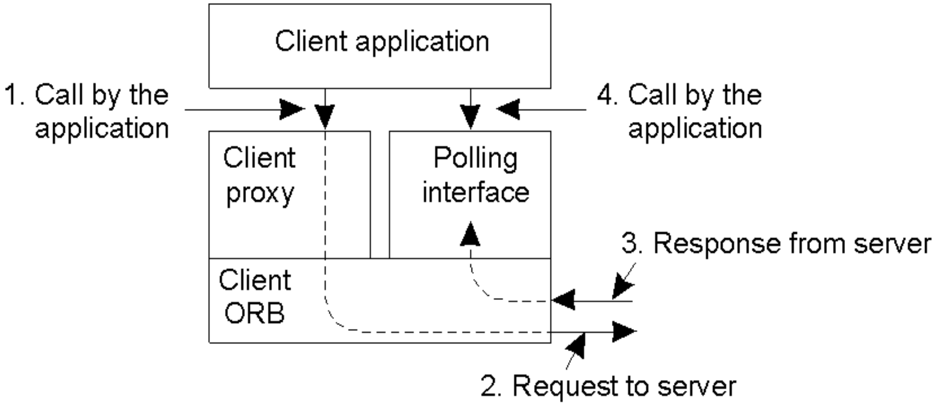
CORBA architecture



Communication in CORBA

- Object invocation models
  - synchronous, asynchronous
- Event and notification services
  - pull, push
- Messaging
  - callback, polling
- Interoperability
  - GIOP (General Inter-ORB), IIOP (Internet Inter-ORB)



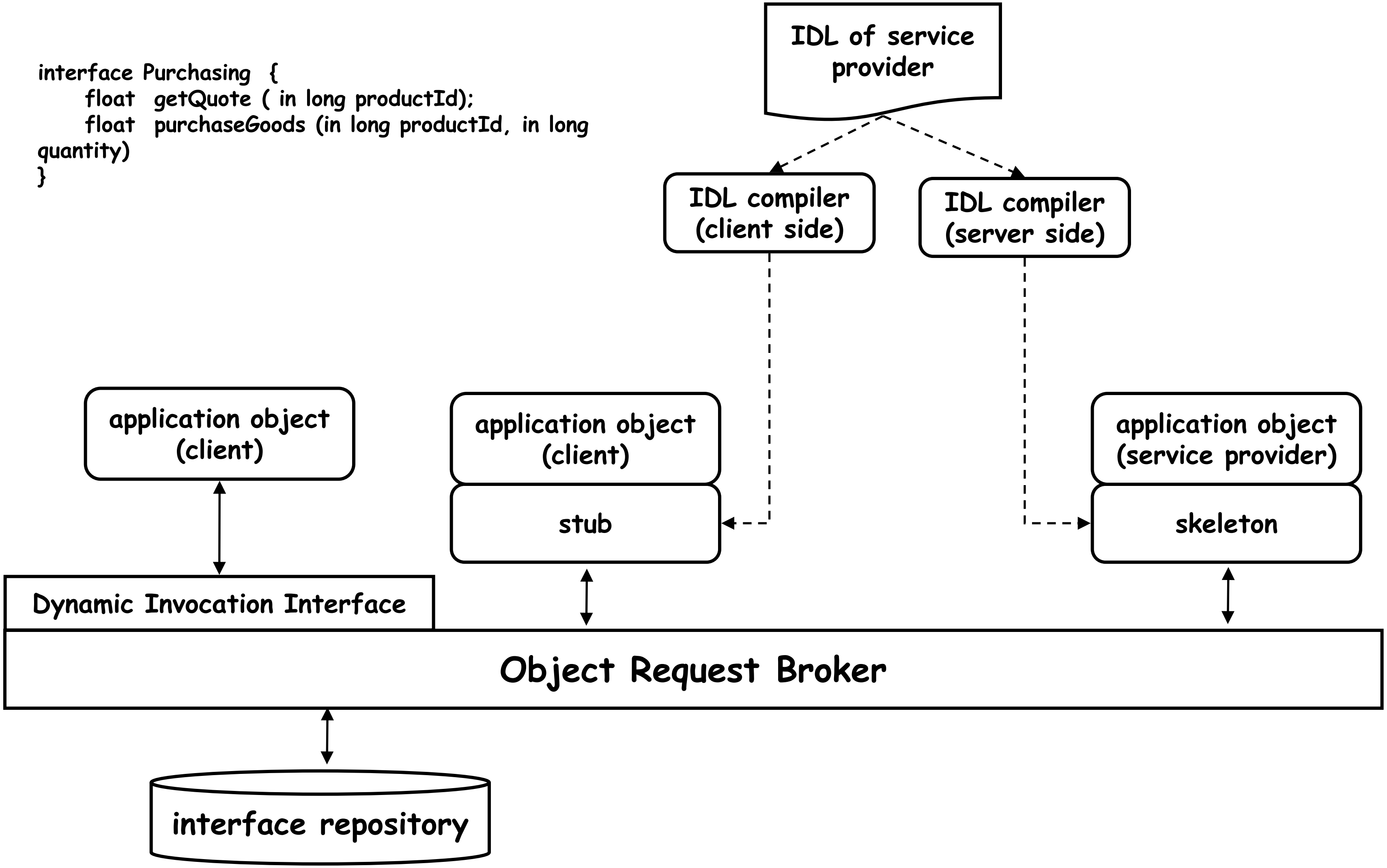


Mechanisms for object invocation

- Object referencing
  - **Simple referencing →** Transparent?
    - network address + endpoint + object id
  - **Location server →** Scalable?
    - **Implementation handle**: a proxy implementation that clients can dynamically download, install, and instantiate when binding to an object
- Static vs. dynamic invocation
  - **Static invocation**
    - The interfaces of an object need to be known when the client application is being developed: e.g. fobject.append(int)
  - **Dynamic invocation**
    - Composes a method invocation at runtime: e.g. invoke(fobject, id(append), int)

How CORBA works

```
interface Purchasing  {
      float  getQuote ( in long productId);
      float  purchaseGoods (in long productId, in long
quantity)
}
```



IDL of service provider

IDL compiler (client side)

IDL compiler (server side)

application object (client)

application object (client)

stub

application object (service provider)

skeleton

Dynamic Invocation Interface

Object Request Broker

interface repository

Common Object Services

Service	Description
Collection	Facilities for grouping objects into lists, queue, sets, etc.
Query	Facilities for querying collections of objects in a declarative manner
Concurrency	Facilities to allow concurrent access to shared objects
Transaction	Flat and nested transactions on method calls over multiple objects
Event	Facilities for asynchronous communication through events
Notification	Advanced facilities for event-based asynchronous communication
Externalization	Facilities for marshaling and unmarshaling of objects
Life cycle	Facilities for creation, deletion, copying, and moving of objects
Licensing	Facilities for attaching a license to an object
Naming	Facilities for systemwide name of objects
Property	Facilities for associating (attribute, value) pairs with objects
Trading	Facilities to publish and find the services on object has to offer
Persistence	Facilities for persistently storing objects
Relationship	Facilities for expressing relationships between objects
Security	Mechanisms for secure channels, authorization, and auditing
Time	Provides the current time within specified error margins

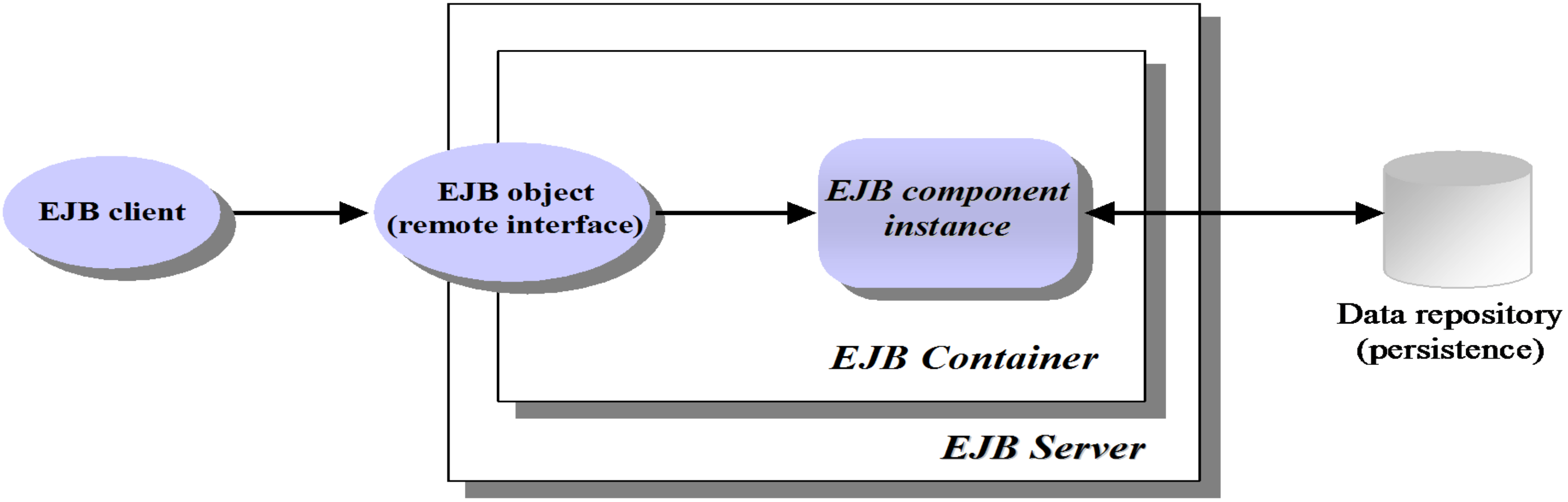
Applications of object-oriented middleware

- Fundamental difference from socket-based messaging
  - The ability to exchange **objects**
- Distributed computing
  - More flexible than RPC
  - Remote computing, edge-based distributed computing
  - Examples
- Application integration via wrapping

The Enterprise Java Beans

- Enterprise Java Beans (EJB) is a server component model for the development and deployment of enterprise-level Java applications based on a distributed object architecture
- EJB Components include:
  - Session Beans
  - Entity Beans
  - Message-Driven Beans

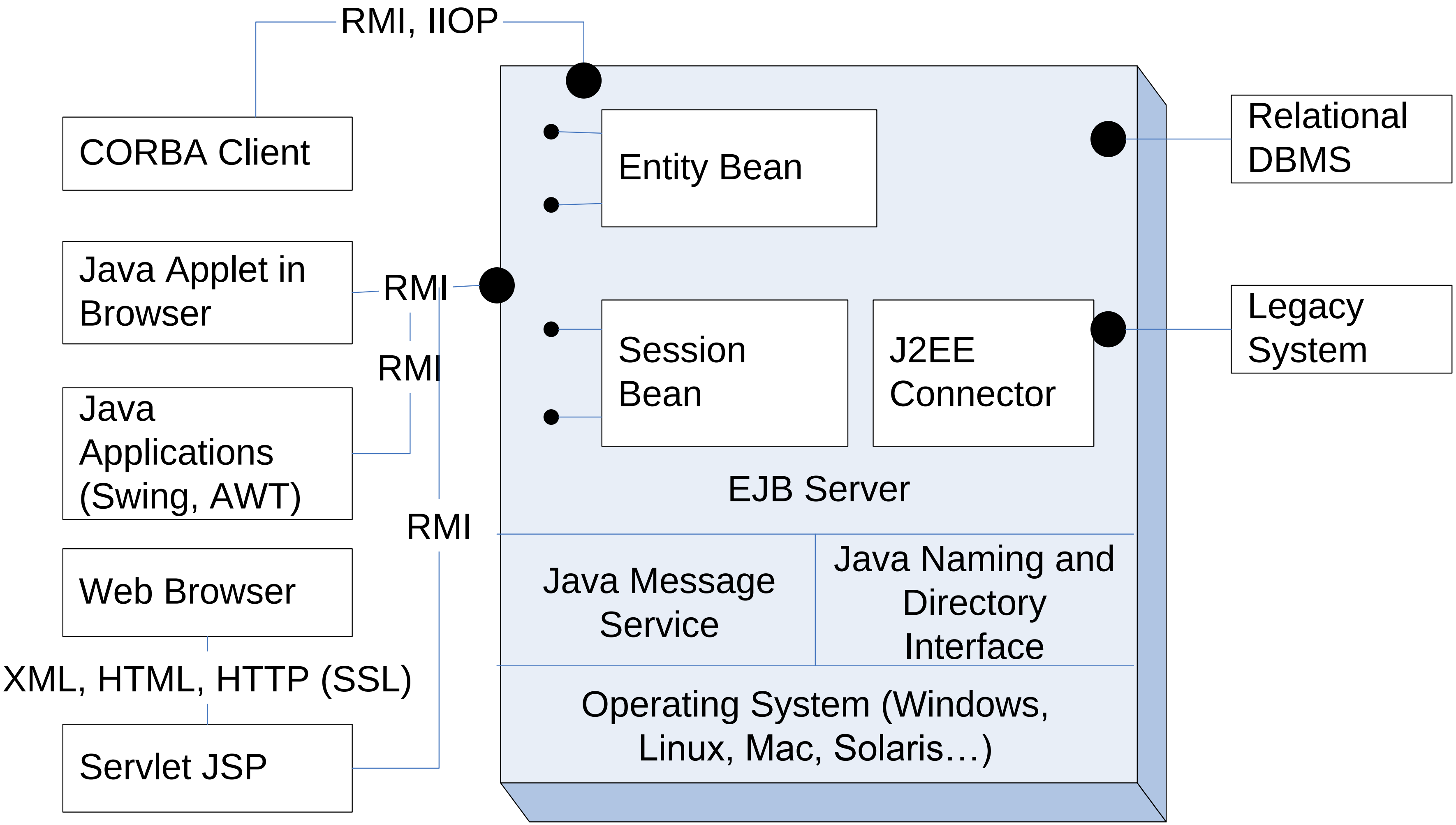
EJBs


EJB client
EJB object
(remote interface)
EJB component
instance
EJB Container
EJB Server
Data repository
(persistence)

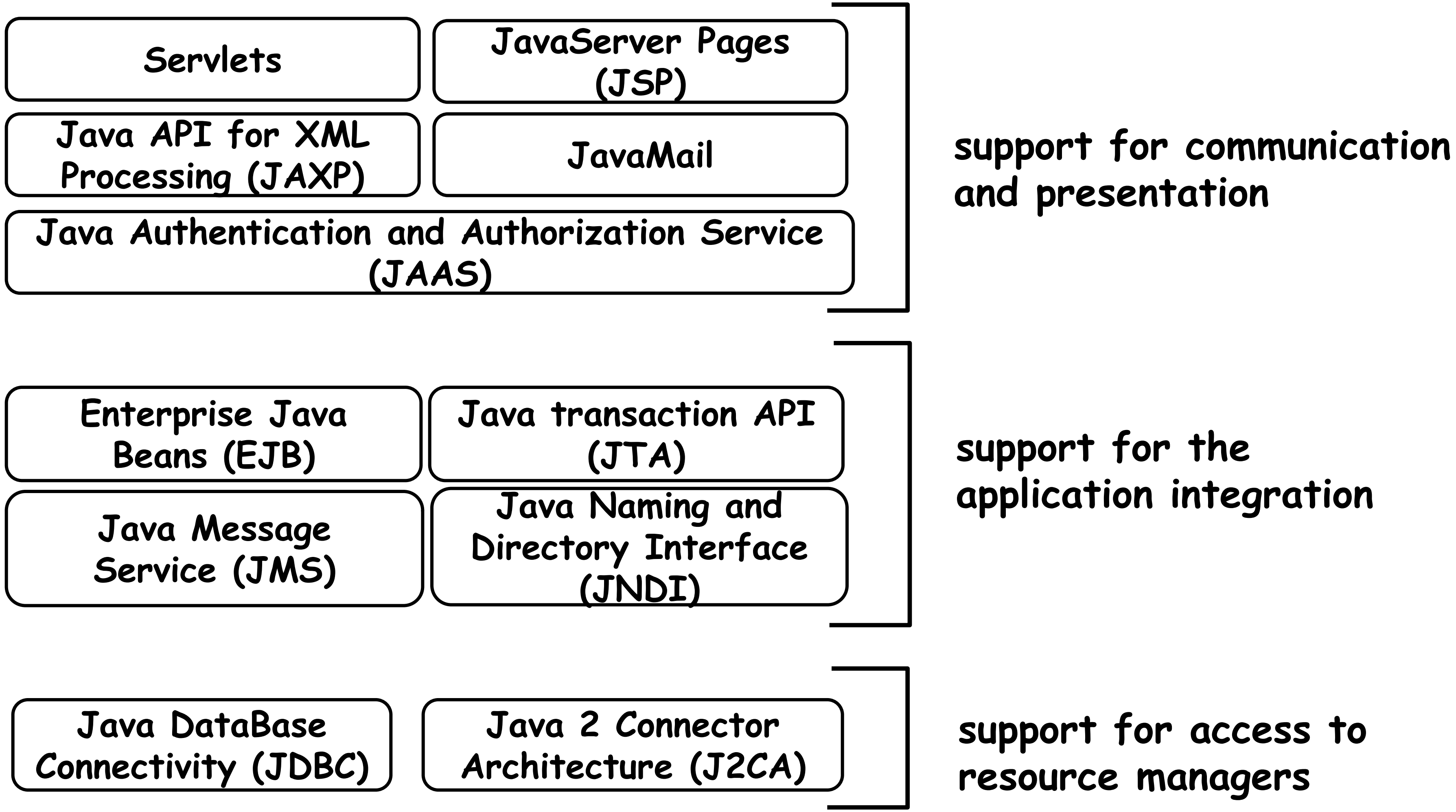
J2EE

- provides a component-based approach to the design, development, assembly, and deployment of enterprise applications
- consists of 3 fundamental parts
  - components
    - a self-contained functional SW unit that is assembled into a J2EE application with its related classes and files and communicates with other components
    - client-tier, web-tier, business-tier, EIS-tier components
  - containers
    - standardized runtime env. that provide application components with specific J2EE system-level services, such as life-cycle management, security, deployment, and runtime services
    - web container, EJB container
  - connectors
    - defines a portable service API that provides access to DB, transaction, naming, directory, and messaging services, and legacy applications
    - JAAS, JAXP, JDBC, JNDI, JTA, JMS, JCA

J2EE Technology



J2EE specifications



Servlets

JavaServer Pages (JSP)

Java API for XML Processing (JAXP)

JavaMail

Java Authentication and Authorization Service (JAAS)

support for communication and presentation

Enterprise Java Beans (EJB)

Java transaction API (JTA)

Java Message Service (JMS)

Java Naming and Directory Interface (JNDI)

support for the application integration

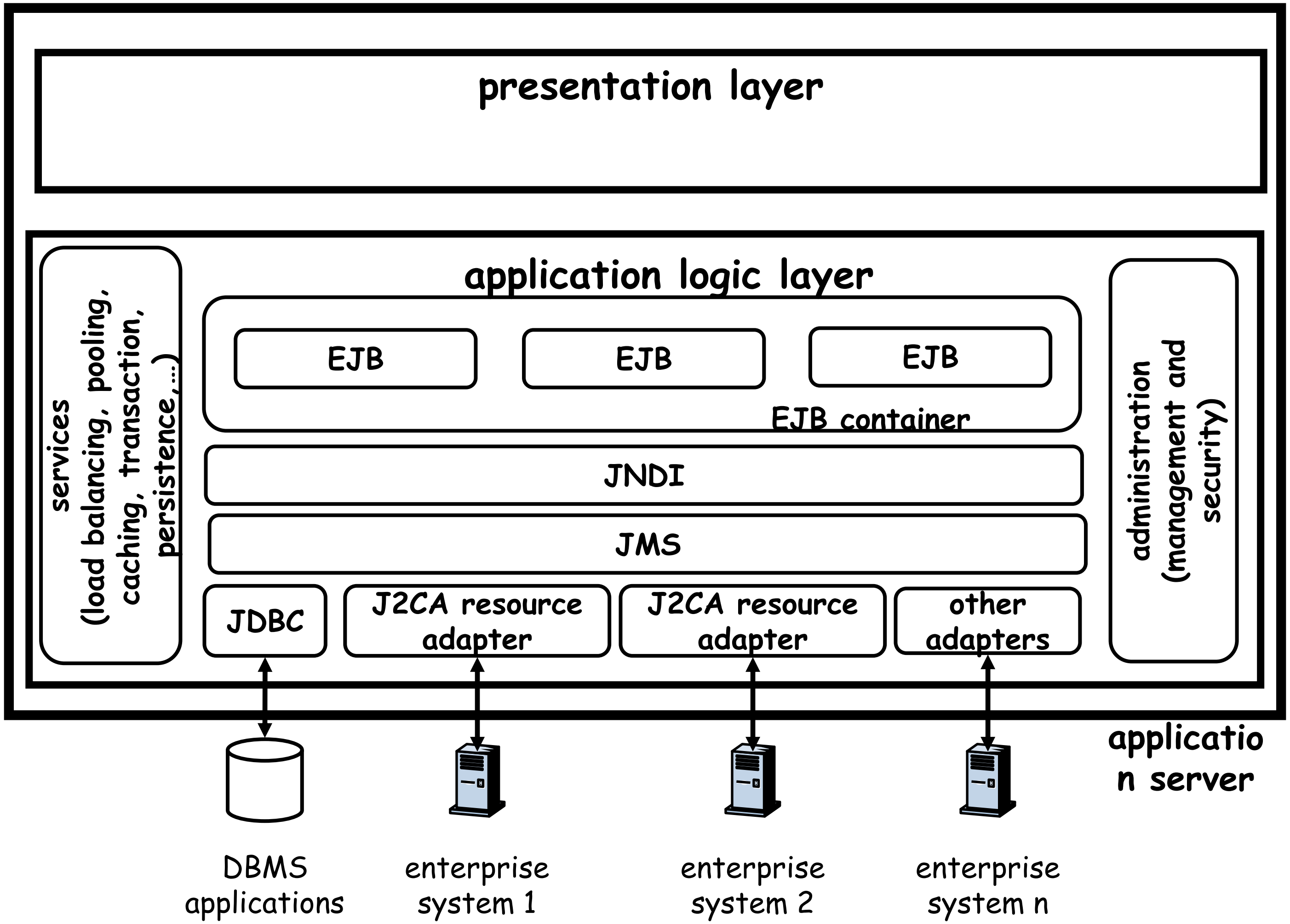
Java DataBase Connectivity (JDBC)

Java 2 Connector Architecture (J2CA)

support for access to resource managers

J2EE

- The support for application logic concentrates on three main specifications: EJB, JNDI, JMS



J2EE

- EJB
  - where the bulk of the application logic resides
  - a server-side component that delivers application-specific functionality such as responding to a request for a quote or processing a purchase order
  - defines 3 different types of beans, based on how they interact with other components and on how they manage state and persistence
- Session beans
  - handle a session with a client
  - e.g., online shopping cart
- Entity beans
  - have a state, stored in a DB or in another persistent storage
- Message-driven beans
  - cater to asynchronous interaction with clients, unlike session or entity beans, which instead interoperate in an RPC-like fashion
  - act as clients to JMS message bus

J2EE

- EJB container
  - provides the environment in which the beans run
  - provides a number of services: transactions, persistence, security
- JNDI
  - defines an interface for directory services, without mandating any implementation
  - clients can bind to servers based on the object name
- JDBC
  - an API that enables developers to access almost any tabular data source by executing SQL commands from a Java program
- J2CA
  - defines how to build resource adapters